Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

David Jerković

Mobilna aplikacija za spremljanje aktivnosti nogometnih ekip

diplomsko delo

univerzitetni študijski program prve stopnje
računalništvo in informatika

Mentor: doc. dr. Damjan Vavpotič

Ljubljana 2015

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Mobilna aplikacija za spremljanje aktivnosti nogometnih ekip

Tematika naloge:

Na trgu je na voljo vrsta mobilnih aplikacij, ki omogočajo merjenje športnih aktivnosti uporabnika mobilne naprave. Namenjene so zlasti posameznikom in ne omogočajo hkratnega merjenja več članov ekipe v skupinskih športih. V okviru diplomske naloge preučite možnosti za uporabo mobilnih naprav za namen hkratnega merjenja več članov nogometnih ekip. Najprej kratko analizirajte obstoječe sorodne aplikacije, nato pa preučite in predstavite ključne tehnologije in rešitve, ki jih je mogoče uporabiti v ta namen. V nadaljevanju diplomske naloge pripravite delujoč prototip aplikacije, ki bo omogočala povezano merjenje članov nogometnih ekip z uporabo mobilnih naprav. Predstavite vse ključne funkcionalnosti, programsko arhitekturo in delovanje prototipa. V zaključku izdelano aplikacijo kritično ovrednotite in podajte smernice za nadaljnje delo.

Izjava o avtorstvu diplomskega dela

Spodaj podpisani David Jerković sem avtor diplomskega dela z naslovom:

Mobilna aplikacija za spremljanje aktivnosti nogometnih ekip

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom doc. dr. Damjana Vavpotiča,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela,
- soglašam z javno objavo elektronske oblike diplomskega dela na svetovnem spletu preko univerzitetnega spletnega arhiva.

V Ljubljani, dne 11. februarja 2014 Podpis avtorja:

Zahvaljujem se družini, ki mi je vedno stala ob strani in me spodbujala.

Za pomoč, nasvete in predloge pri izdelavi diplomskega dela se iskreno zahvaljujem mentorju doc. dr. Damjanu Vavpotiču.

Najlepša hvala moji punci Tini, ki me je vsa leta študija prenašala in mi neprestano pomagala. Brez njene pomoči mi ne bi uspelo.

Kazalo

Povzetek

Abstract

1 Uvod **1**

2 Analiza obstoječih aplikacij **3**

3 Uporabljene tehnologije in orodja **7**

3.1 Strežniški del 7

3.2 Mobilna aplikacija 8

4 Arhitektura in razvoj informacijskega sistema **15**

4.1 Strežniški del 18

4.2 Mobilna aplikacija 20

5 Sklepne ugotovitve **43**

Literatura **47**

Slike

3.1 Diagram komponent GCM 11

4.1 Diagram UML uporabe aplikacije 16
4.2 Diagram komponent informacijskega sistema 17
4.3 Slika podatkovne baze 19
4.4 Domači pogled aplikacije 20
4.5 Trenutno gibanje igralca 28
4.6 Pregled trenutnega gibanja igralcev 30
4.7 Prikaz pregleda aktivnosti igralca 37

Seznam uporabljenih kratic

kratica	angleško	slovensko
GPS	Global Positioning System	sistem globalnega pozicioniranja
SQL	Structured Query Language	strukturirani poizvedovalni jezik
AMP	Apache, MySQL, Perl/PHP/Python	skupek programskih komponent
XAMPP	cross-platform AMP	večplatformni AMP
SDK	software development kit	zbirka orodij za razvoj programske opreme
ADT	Android Development Tools	zbirka orodij za razvoj Android aplikacij
API	application programming interface	vmesnik za programiranje aplikacij
GCM	Google Cloud Messaging	storitev, ki omogoča prenos sporočil med Android napravami
JDBC	Java Database Connectivity	tehnologija, ki omogoča povezavo do podatkovnih baz
UML	Unified Modeling Language	poenoteni jezik modeliranja
JSON	JavaScript Object Notation	besedilna oblika zapisa za izmenjavo podatkov
SHA-1	cryptographic hash function	kriptografska zgoščevalna funkcija

Povzetek

Večina ekip v moštvenih športih bi potrebovala preprost in lahko dostopen informacijski sistem, ki bi hranil informacije o njenih članih ter aktivnostih, ki jih izvajajo.

Odločili smo se razviti nizkocenovno rešitev v obliki mobilne aplikacije za platformo Android, ki člane ekipe združuje v skupine in jim ponuja ločene funkcionalnosti. Igralcem prek storitve Google Location omogoča pridobivanje informacij o gibanju med treningom, trenerjem pa nudi takojšen pregled nad prejetimi podatki. Služi jim kot orodje za upravljanje moštva ter analizo učinkovitosti treningov.

V diplomskem delu smo analizirali obstoječe rešitve in jih primerjali z našo. Nato smo predstavili uporabljene tehnologije in opisali zgradbo našega sistema. V osrednjem delu smo se lotili razvoja aplikacije in njenega izgleda ter prikazali implementacijo opisanih tehnologij. Na koncu smo omenili možne izboljšave izdelka ter morebitne razširitve.

Ključne besede: Android, MySQL, Location Services, Google Cloud Messaging, Google Maps, mobilne aplikacije, nogomet.

Abstract

Most sports teams are in need of an easily accessible information system which stores information about its members and activities they carry out.

We have decided to develop a low-cost solution in the form of a mobile application for the Android platform. The application groups members of the team and offers them separate functionalities. It gives players access to information about their movement during exercises via Google Location Services and provides coaches with a way of obtaining latest collected data in real-time. The application serves as a team management and training analysis tool.

The thesis begins with analyzing the existing solutions and comparing it to our own. It contains the presentation of the technology used in development and describes the structure of our system. The central part of our thesis explains the development process of the application and shows the implementation of the described technologies. Lastly, it mentions several improvements of the product as well as possible extensions. too short.

Keywords: Android, MySQL, Location Services, Google Cloud Messaging, Google Maps, mobile applications, football.

Poglavje 1

Uvod

Mobilne aplikacije uporabljamo vsakodnevno in na različnih področjih, med drugim tudi v športu. Trgovina Google Play je polna aplikacij, namenjenih spremljanju uporabnikov med tekom in nudenjem informacij o njihovi aktivnosti, medtem ko primanjkuje aplikacij, ki bi bile namenjene organizaciji in upravljanju v ekipnih športih. Trenerji se pri sestavi treningov še vedno držijo lastnih zapiskov, taktiko pa načrtujejo na papirju, risalni tabli ali doma na računalniku. Tudi obveščanje in komunikacija med trenerji in igralci še vedno poteka prek klicev, sporočil in elektronske pošte. Trenerjem v manjših klubih prav tako primanjkuje dodatnih pripomočkov, kot so pasovi za merjenje utripa, s katerimi bi lahko spremljali učinkovitost igralcev med samim treningom. Ekipe v moštvenih športih zaenkrat nimajo pravega informacijskega sistema, ki bi trenerjem olajšal delo ter združeval zgoraj naštete dejavnosti.

Naš cilj je bil ustvariti nizkocenovno mobilno aplikacijo, ki bi ekipam ponujala informacijski sistem za upravljanje in organizacijo dejavnosti znotraj ekipe. Trenerjem bi omogočala pripravo treningov ter spremljanje gibanja igralcev med samo aktivnostjo prek različnih storitev, ki jih ponuja Google. Na podlagi zbranih podatkov bi imeli trenerji na razpolago pregled zgodovine aktivnosti igralcev ter različna orodja za njihovo analizo. Prek storitve Google Cloud Messaging bi aplikacija hkrati nudila tudi enostavno in učinkovito

rešitev za komunikacijo med trenerjem in igralci.

V diplomskem delu smo najprej predstavili analizo najbolj priljubljenih konkurenčnih aplikacij ter izpostavili razlike med njimi in našo rešitvijo. Nato smo na kratko pregledali tehnologije in metode, ki smo jih uporabili pri razvoju projekta. V četrtem poglavju smo opisali arhitekturo in razvoj informacijskega sistema ter izpostavili najzanimivejše funkcionalnosti aplikacije. Na koncu smo podali še sklepne ugotovitve diplomskega dela.

Poglavje 2

Analiza obstoječih aplikacij

Na trgu obstaja ogromno aplikacij, s katerimi lahko merimo športno aktivnost uporabnikov. Obstoječe rešitve merijo lastnosti gibanja, kot so pretečena razdalja, trenutna lokacija in hitrost. Predvsem so namenjene spremljanju lastnih rezultatov opravljenih aktivnosti. Primanjkuje pa aplikacij, ki bi bile namenjene ekipam v moštvenih športih in imele možnost spremljanja podobnih spremenljivk. Na trgu lahko zasledimo aplikacije z uporabniškim vmesnikom, ki omogočajo risanje treningov in nadomeščajo papir ali risalno tablo. Našli pa smo tudi aplikacijo, v katero si lahko shranjujemo podatke o ekipi in njenih igralcih ter urejamo podrobnosti dogodkov, na katerih so bili člani ekipe prisotni. Naša aplikacija združuje oba tipa aplikacij. Omogoča spremljanje lastne učinkovitosti igralcev na treningih, trenerjem pa nudi celoten pregled nad trenutno aktivnostjo ter analizo pridobljenih podatkov. Spodnja tabela primerja glavne značilnosti najbolj znanih aplikacij v primerjavi z našo.

	Tekaške aplikacije	**Soccer Dad**	**Coacher**
Tipi uporabnikov	športniki	trenerji	trenerji in športniki
Namen uporabe	merjenje lastne aktivnosti	upravljanje lastne ekipe	merjenje lastne aktivnosti in upravljanje ekipe
Vnos podatkov	samodejno merjenje lastne pretečene razdalje in hitrosti	ročni vnos podatkov ekipe in njenih članov ter ročno vpisovanje podrobnosti aktivnosti	samodejno merjenje pretečene razdalje in hitrosti igralcev
Interakcija med uporabniki	primerjava rezultatov in možnost objave dosežkov na družbenih medijih	deljenje treningov z ostalimi uporabniki, pošiljanje elektronske pošte in tekstovnih sporočil članom skupine	primerjava rezultatov posameznih igralcev, pregled rezultatov celotne ekipe in pošiljanje obvestil igralcem

Tabela 2.1: Tabela primerjav najpomembnejših funkcionalnosti

Najbolj znane tekaške aplikacije, kot so Runtastic, Runkeeper, Nike+ Running in iSmooth Pro, so namenjene merjenju uporabnikove trenutne pretečene razdalje in hitrosti ter prikazovanju pridobljenih podatkov v danem trenutku. Omenjene aplikacije nudijo pregled nad shranjenimi aktivnostmi s prikazi najpomembnejših podatkov, kot so povprečna in najvišja hitrost, pretečena razdalja ter tempo teka, ter omogočajo objavljanje dosežkov prek različnih družbenih omrežij in povezovanje s prijatelji. Nekatere aplikacije imajo možnost povezovanja s posebnimi napravami, ki oddajajo nizkoenergijski signal Bluetooth. Uporabne podatke lahko pridobivajo prek naprav, kot so pasovi za merjenje srčnega utripa in ure s signalom GPS. Brezplačne verzije večine aplikacij imajo omogočene zgoraj naštete funkcionalnosti, medtem ko plačljive različice nudijo dodatke, kot so prilagojen načrt vadb, glasovno spremljanje osebnega trenerja ter motivacija pri teku. Najbolj znane tekaške aplikacije se razlikujejo v manjših podrobnostih. RunKeeper in Nike+ Running sta izkušnjo nadgradila z različnimi izzivi, v katerih lahko uporabnik tekmuje s prijatelji in z njimi primerja dosežke. iSmooth Pro vsebuje funkcijo, imenovano Ghost Run. Trenutno aktivnost primerja z najboljšim dosežkom na isti poti in ju izrisuje istočasno.

Aplikacija Soccer Dad je upravljalni program za naprave iOS, ki olajša delo trenerjem nogometnih ekip. Nudi popoln pregled nad člani moštva in organizacijo vseh podatkov ekipe. Omogoča načrtovanje treningov in tekem, integracijo ustvarjenih dogodkov s koledarjem iOS ter pošiljanje obvestil članom moštva prek elektronskih in tekstovnih sporočil. Med treningom in samo tekmo ima trener pregled nad rezervami in trenutno postavo, ki jo lahko poljubno spreminja. Ročno lahko vnaša dogodke na tekmi ter si zapisuje statistiko igralcev, na podlagi katere lahko kasneje dela različne analize. Ena od zanimivejših funkcionalnosti aplikacije Soccer Dad je uporabniški vmesnik, ki spominja na risalno ploščo, s pomočjo katere lahko uporabnik riše, načrtuje in prikazuje taktiko ter posamezne vadbe na treningih. Taktiko in treninge si lahko tudi shrani v napravi in deli z drugimi uporabniki aplikacij.

Našo aplikacijo, imenovano Coacher, lahko za razliko od tekaških aplikacij in aplikacije Soccer Dad uporabljata dva tipa uporabnikov. Trenerji in igralci so združeni v skupino (nogometni klub), kjer ima trener popoln pregled nad podatki ekipe, posameznimi igralci ter aktivnostmi, ki so jih izvajali. Igralcem, ki so pri aktivnosti, aplikacija meri pretečeno razdaljo in hitrost ter pridobljene podatke v danem trenutku tudi prikazuje. Podatke pošilja na strežnik, trenerju pa vrača celoten pregled nad trenutno aktivnostjo udeleženih igralcev. Uporabniki si lahko pridobljene podatke kasneje tudi ogledajo in iz njih izluščijo najpomembnejše informacije. Trener lahko igralce prek kratkih sporočil kadarkoli obvešča o morebitnih spremembah pri aktivnostih ter jim daje ustrezne napotke.

Coacher na nek način torej združuje tekaške aplikacije z upravljanjem nogometnih ekip. Trenerjem omogoča poceni možnost natančnega merjenja učinkovitosti igralcev med samim treningom. Aplikacija lahko na podlagi statistik in analiz pridobljenih podatkov pripomore k izboljšanju pripravljenosti igralcev ter prilagajanju in oblikovanju ustreznih treningov. Prav tako omogoča popoln pregled nad administracijo celotne ekipe in igralcev. Tekaške aplikacije merijo gibanje uporabnika in podatke shranjujejo v telefon, medtem ko moramo v aplikaciji Soccer Dad podatke o ekipi ročno vnašati v napravo. Coacher združuje funkcionalnosti obeh vrst aplikacij in podatke pošilja na podatkovni strežnik, do katerega lahko dostopajo tako igralci kot trener v ekipi.

Poglavje 3

Uporabljene tehnologije in orodja

3.1 Strežniški del

3.1.1 MySQL

MySQL je eden izmed največjih odprtokodnih sistemov za upravljanje relacijskih podatkovnih baz. Ponuja hitro, enostavno in zanesljivo rešitev shranjevanja in urejanja podatkov ter poizvedovanja po njih. Uporabljajo ga nekatere izmed največjih spletnih strani na svetu, kot so Facebook, Google, YouTube in Twitter. Najbolj znana brezplačna orodja za upravljanje podatkovnih baz MySQL so MySQL Workbench, phpMyAdmin in Webmin.[1]

3.1.2 MySQL Workbench

Pri razvoju naše aplikacije smo uporabili uradno orodje podjetja Oracle, in sicer MySQL Workbench, ki omogoča upravljanje podatkovnih baz MySQL. Gre za zelo močno orodje, ki skrbi za povezavo do podatkovnih baz ter urejanje pravic in dostopov do nje. Prav tako nudi možnost modeliranja same podatkovne baze, ustvarjanje entitetno-relacijskih diagramov ter shranjevanje, brisanje, spreminjanje podatkov in poizvedovanje po njih. Poleg urejanja

nudi tudi različna orodja za opazovanje delovanja podatkovne baze, spremlja učinkovitost dostopov, promet na strežniku, število povezav do baze, količino bralnih in pisalnih operacij itd.[2]

3.1.3 XAMPP

MySQL je osrednja komponenta orodij AMP, ki omogočajo vzpostavitev in delovanje strežnika ter skrbijo za pravilno izvajanje dinamičnih spletnih strani.

XAMPP je večplatformni skupek programskih komponent, sestavljen iz strežnika Apache HTTP, podatkovne baze MySQL in programskih tolmačev, napisanih v programskih jezikih PHP in PERL. Namenjen je razvoju in testiranju spletnih strani in aplikacij na lastnih računalnikih. Razvijalcem nudi strežniško okolje, prek katerega lahko dostopajo do podatkov na podatkovnih bazah, določajo, kdo lahko do njih dostopa, ter upravljajo s konfiguracijo strežnika.[3]

3.2 Mobilna aplikacija

3.2.1 Android

Android je odprtokodni mobilni operacijski sistem, ki temelji na Linuxovem jedru. Sprva je bil specializiran za mobilne naprave in tablice z zaslonom na dotik, kasneje pa se je razširil tudi na igralne konzole, televizije, pametne ure, očala in celo avtomobile. Android je v lasti Googla, ki skrbi za njegov nadaljnji razvoj in distribucijo.[4]

Zaradi svoje odprtokodnosti je izjemno prilagodljiv sistem, ki mu ogromno število razvijalcev konstantno dodaja nove funkcionalnosti. Android ločuje strojno opremo od programske, ki teče na napravi ter omogoča enostaven in hiter razvoj aplikacij. Ena od prednosti platforme Android je ustvarjanje aplikacij, ki se na različnih napravah obnašajo enako. V zadnjih letih je prav zato uporaba Androida močno narasla.

3.2.2 Razvojno okolje

Razvojno okolje, uporabljeno za razvoj naše aplikacije, je sestavljeno iz programskega okolja Eclipse, orodij Android SDK in vstavka za Ecplise ADT.[5]

Android Software Developer Kit je zbirka razvojnih orodij, potrebnih za razvoj Android aplikacij. Vsebuje razhroščevalnik in različne programske knjižnice, emulator, potrebno dokumentacijo in osnovne primere z navodili.[6]

Programsko okolje Eclipse je brezplačno, odprtokodno razvojno orodje, napisano v Javi. V osnovni različici vsebuje razvojno okolje za Javo, z različnimi vtičniki pa omogoča razvoj aplikacij v nekaterih drugih programskih jezikih, kot so PHP, C, C++, C# itd. Do prihoda Androida verzije 5 in z njim orodja Android Studio je bil Eclipse tudi uradno orodje za razvoj aplikacij za Android.[7]

Vstavek ADT vgradi v Eclipse različna orodja, s katerimi poteka razvoj Android aplikacij hitreje in enostavnejše. ADT integrira enostavno ustvarjanje novih in uvažanje obstoječih projektov, prav tako pa omogoča grajenje programa in skrbi za njegovo izvajanje. Vsebuje tudi obsežno dokumentacijo kode in preverjevalnik sintakse, v Eclipse pa doda tudi XML urejevalnike in grafične vmesnike za oblikovanje, s katerimi olajša urejanje uporabniškega vmesnika aplikacije.[8]

3.2.3 Storitve Google Play

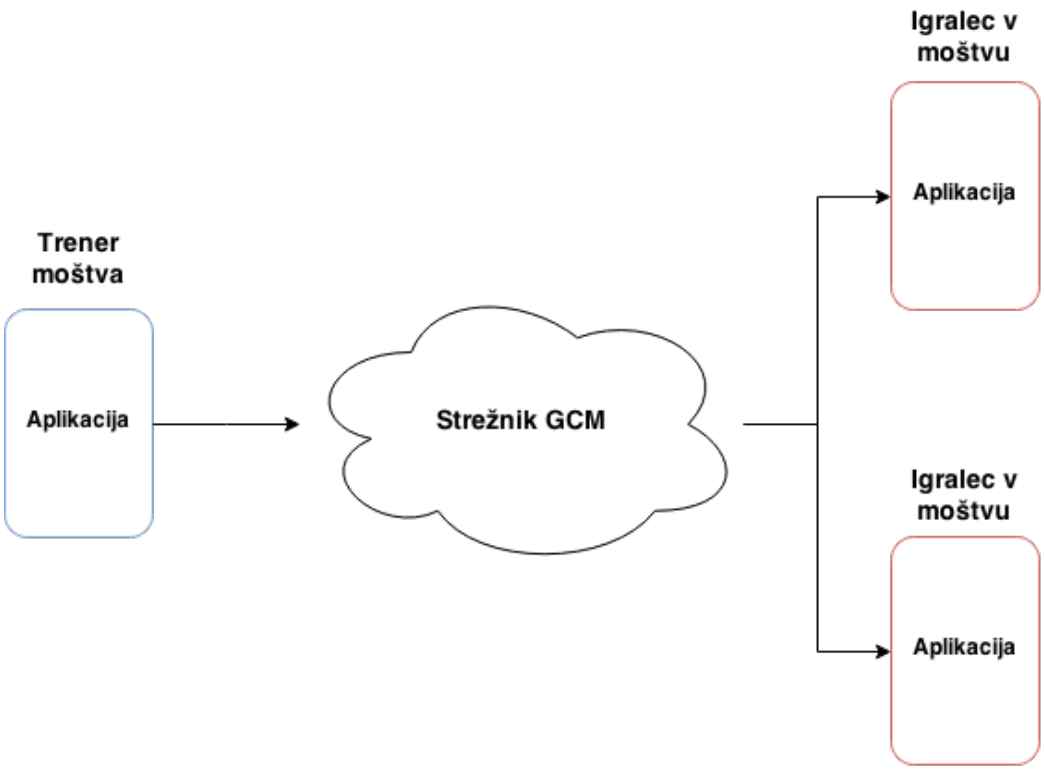
Storitve Google Play omogočajo mobilnim aplikacijam dostop do najnovejših storitev, ki jih ponuja Google. Zaradi tesne povezave z operacijskim sistemom Android in zmogljivimi programskimi knjižnicami omogočajo razvijalcem enostaven in hiter razvoj dodatnih funkcionalnosti, ki jih ponujajo. Povezovanje poteka prek enotnega avtorizacijskega API-ja, ki Google račun povezuje z ustrezno napravo in prek avtorizacijskih žetonov nudi varno in zanesljivo povezavo do storitev. Storitve Google Play so združene v APK – Android Application Package in objavljene na trgovini Google Play. Ta poskrbi za ustrezno namestitev storitev in njihovo avtomatsko posodabljanje. Razvijalci lahko tako brezskrbno razvijajo aplikacije, ki uporabljajo omenjene storitve in se ne obremenjujejo s kompatibilnostjo naprav uporabnikov.

Med najbolj znane storitve Google spadajo Google Maps API, družbeno omrežje Google+, storitve Location, oblačna storitev za shranjevanje podatkov Drive, digitalna denarnica Wallet, Google Cloud Messaging API itd.[9]

3.2.4 Google Cloud Messaging

Google Cloud Messaging je brezplačna storitev, ki razvijalcem omogoča pošiljanje podatkov z lastnih strežnikov na naprave Android. Namenjena je obveščanju naprav o spremembah na strežniku in pošiljanju kratkih sporočil.

GCM aplikaciji podatke samo pošilja, njihovo obdelavo pa prepušča aplikaciji. Pri samem prejemanju podatkov aplikaciji ni treba teči, saj GCM prek ustreznih odjemalcev poskrbi, da se naprava zbudi in prejme podatke. Za pravilno delovanje moramo pravilno implementirati odjemalce ter od naprave zahtevati določena dovoljenja.[10]



Slika 3.1: Diagram komponent GCM

Trenerji s pomočjo podatkovne baze in podatkov igralcev ustvarijo sporočila ter jih prek zahteve HTTP POST pošljejo strežniku GCM.

Strežniki GCM prevzemajo sporočila trenerjev prek HTTP POST zahtev in jih hranijo pri sebi. Sporočila čakajo v vrsti, dokler se naprave uporabnikov ne povežejo na internet, nato pa jih strežnik dostavi.

Uporabniki se pri vpisu v aplikacijo istočasno registrirajo tudi na storitev GCM. Pri uspešni registraciji prejmejo ključ RegistrationID, ki si ga shranijo v podatkovno bazo. Prek tega ključa lahko nato strežnik GCM ustreznim napravam dostavi prejeto sporočilo.

3.2.5 Location Service API

Location Service API aplikacijam omogoča pridobivanje informacij o položaju naprave. Po uspešni vzpostavitvi povezave lahko API v aplikaciji spremlja spremembo lokacije, trenutno hitrost ter jih primerja s prejšnjimi podatki. Vračanje informacij o trenutnem položaju lahko razvijalci z različnimi nastavitvami prilagodimo našim potrebam.

Poleg določanja trenutnega položaja Location Service API omogoča tudi prepoznavanje tipa gibanja prek storitve Activity Recognition. Na podoben način, kot deluje pridobivanje informacij o položaju naprave, lahko s to storitvijo ugotovimo, ali se uporabnik giba ali miruje. Ena od zanimivejših funkcionalnosti API-ja je tudi Geofencing, ki spremlja bližino naprave aplikaciji zanimivih lokacij. Geofence je objekt, ki vsebuje geografsko širino in dolžino ter polmer, na podlagi katerega lahko definiramo oddaljenost lokacije. Geofence tako deluje kot območje, v katerega lahko uporabnik vstopi in izstopi.[11]

Location Service API je naslednik Android Framework Location API-ja ter pomemben del storitev Google Play. V primerjavi s predhodnikom samodejno poskrbi za prejemanje podatkov glede na trenutni način povezave. Razvijalcem ni več treba ročno nastaviti tipa prejemanja glede na brezžično omrežje ter signal GPS. Prav tako podrobneje spremlja gibanje naprave in nudi boljšo natančnost določanja položaja. Razvijalci si lahko prilagodimo stopnjo natačnosti delovanja glede na porabo baterije, v večini primerov pa storitev v primerjavi z Android Framework Location API-jem omogoča večjo natančnost z manjšo porabo energije.

3.2.6 Google Maps Android API v2

Google Maps Android API v2 omogoča vgrajevanje zemljevidov Google v Android aplikacijo. Prednost API-ja je v tem, da samodejno poskrbi za povezavo do strežnikov Google Maps, prenašanje podatkov, prikazovanje zemljevidov ter upravljanje uporabnikove interakcije z zemljevidom. Uporabniki se lahko prek API-ja po zemljevidu premikajo, na zemljevid postavljajo točke, rišejo črte, prikazujejo informacije o položaju itd.

Za pravilno delovanje API-ja v naši aplikaciji potrebujemo Android SDK ter ustrezen sklic na storitve Google Play. Pridobiti moramo še ustrezen Maps API ključ, ki je nujen za povezavo s strežniki Google Maps. Poleg ključa je v aplikaciji treba zahtevati ustrezna dovoljenja od naprave ter dovoliti uporabo OpenGL ES verzije 2, ki omogoča izris zemljevida.[12]

3.2.7 Graphview

Programska knjižnica GraphView omogoča enostavno implementacijo preglednih in prilagodljivih grafov v Android aplikacijo. Razvijalci si lahko prilagodijo nastavitve, kot so prikaz legende, izpis raznih naslovov in pripisov ter izgled samega grafa. GraphView omogoča tudi risanje večih grafov v istem koordinatnem sistemu ter samodejno implementacijo odzivov na uporabnikove geste, kot so povečava in pomikanje po grafu.[13]

3.2.8 JDBC

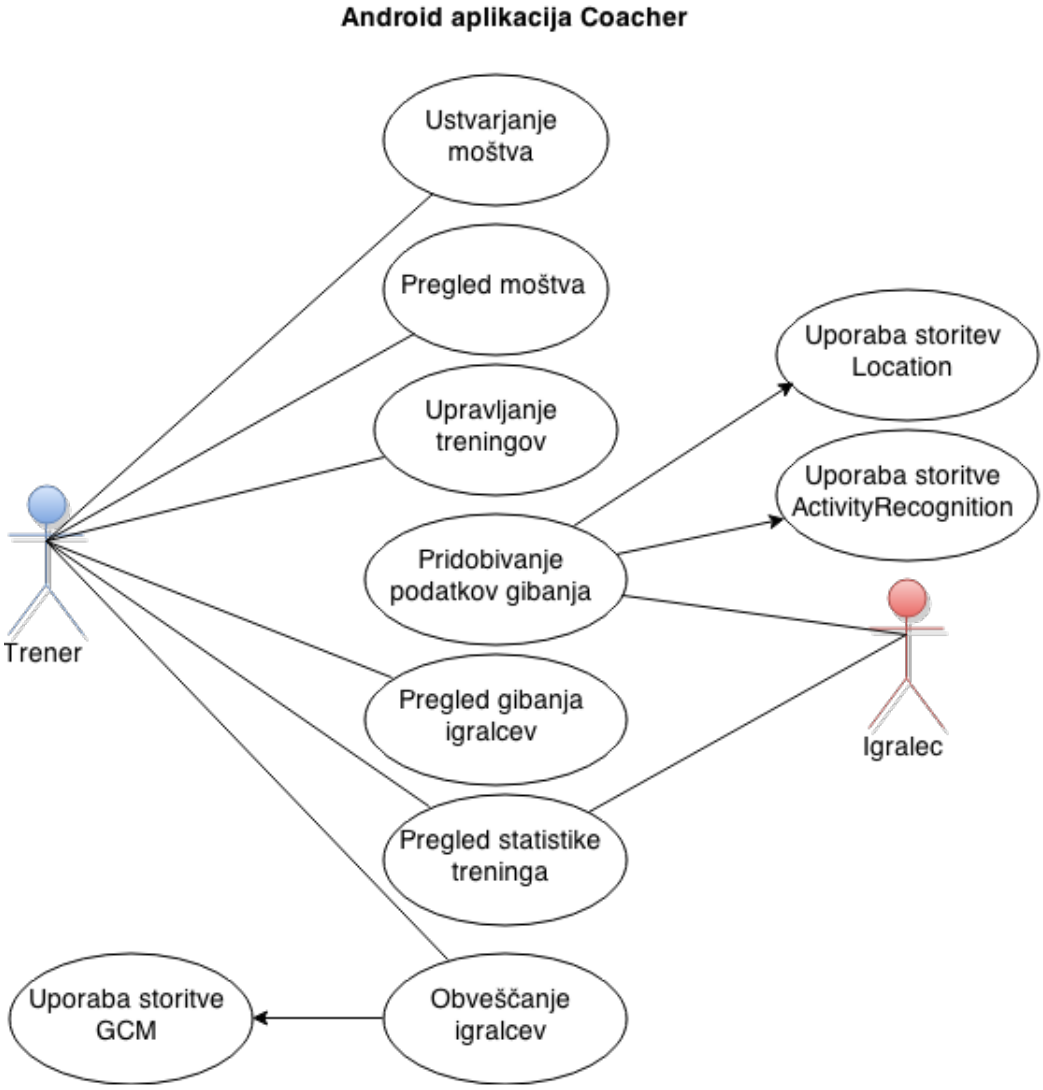
Java Database Connectivity je API, spisan v Javi, ki odjemalcu omogoča dostop do podatkovne baze SQL. Namenjen je izvajanju operacij nad relacijskimi podatkovnimi bazami. JDBC nudi sredstva za poizvedovanje po podatkovnih bazah ter funkcije za urejanje, dodajanje in brisanje podatkov v njej ter gonilnike za Java programe in Android aplikacije. S pomočjo JDBC Android API-ja lahko enostavno in hitro vzpostavimo povezavo med aplikacijo in podatkovno bazo. Po uspešni povezavi pa nam API omogoča pošiljanje poizvedb in obdelovanje pridobljenih rezultatov.[14]

Poglavje 4

Arhitektura in razvoj informacijskega sistema

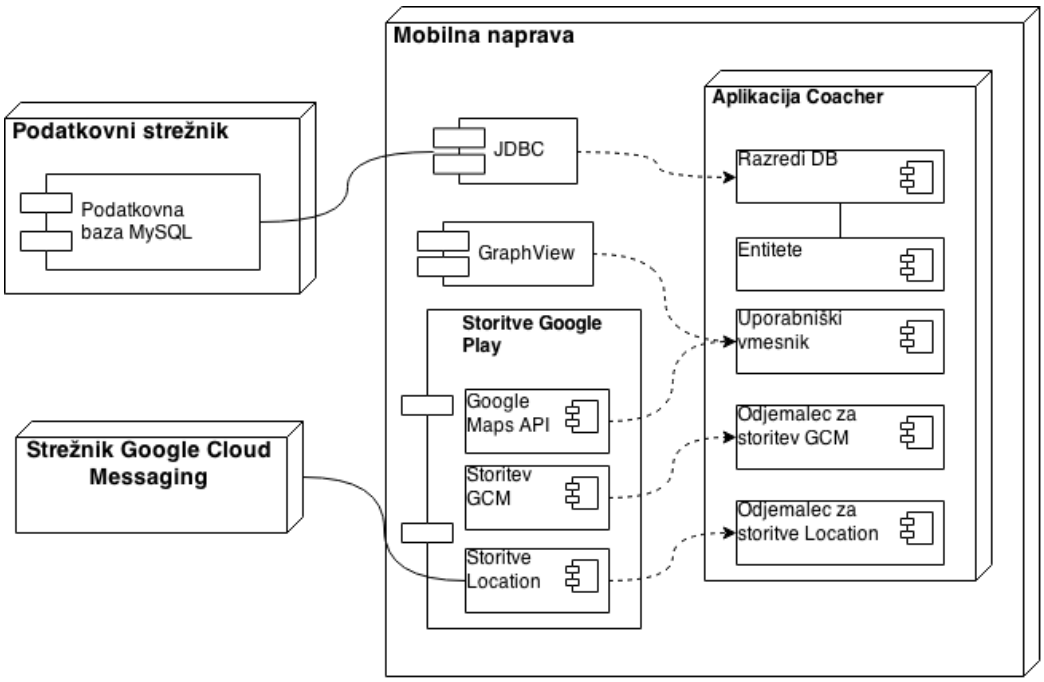
V nadaljevanju poglavja bomo opisali arhitekturo in delovanje informacijskega sistema ter predstavili najzanimivejše segmente rešitve. Ogledali si bomo posamezne komponente sistema, povezave med njimi in prikazali implementacijo opisanih tehnologij.

Aplikacijo uporabljata 2 tipa uporabnikov; trenerji in igralci. Vsak uporabnik ima možnost postati trener, ustvariti skupino, ki predstavlja moštvo v ekipnih športih, ter določiti njene člane. Uporabniki s tem pridobijo popoln pregled nad podatki moštva, igralci in aktivnostmi, ki so se v njem izvajale. V svojih moštvih imajo možnost ustvariti treninge, ki so vidni vsem članom moštva. O začetku treninga, koncu ali drugih spremembah lahko igralce prek aplikacije tudi obvestijo. Ob začetku treninga se mu igralci pridružijo in pričnejo z zbiranjem podatkov. Med samim treningom imajo trenerji popoln pregled nad trenutno aktivnostjo vseh igralcev, prek kratkih sporočil pa jim lahko pošljejo tudi napotke. Po končanem treningu si lahko ogledajo poročilo treninga ter celotno aktivnost posameznih igralcev.



Slika 4.1: Diagram UML uporabe aplikacije

Igralci lahko zaprosijo za pridružitev skupini ali se pa se ji pridružijo prek trenerjevega povabila. Na domačem zaslonu imajo pregled nad svojimi podatki, podatki moštva ter prihajajočimi, trenutnimi in končanimi treningi. Po pridružitvi na izvajajoči trening aplikacija prične z zbiranjem podatkov. Ti podatki se interaktivno prikazujejo, hkrati pa shranjujejo na strežnik. Po končanem treningu imajo igralci, prav tako kot trener, pregled nad aktivnostjo in njenimi najpomembnejšimi informacijami.



Slika 4.2: Diagram komponent informacijskega sistema

4.1 Strežniški del

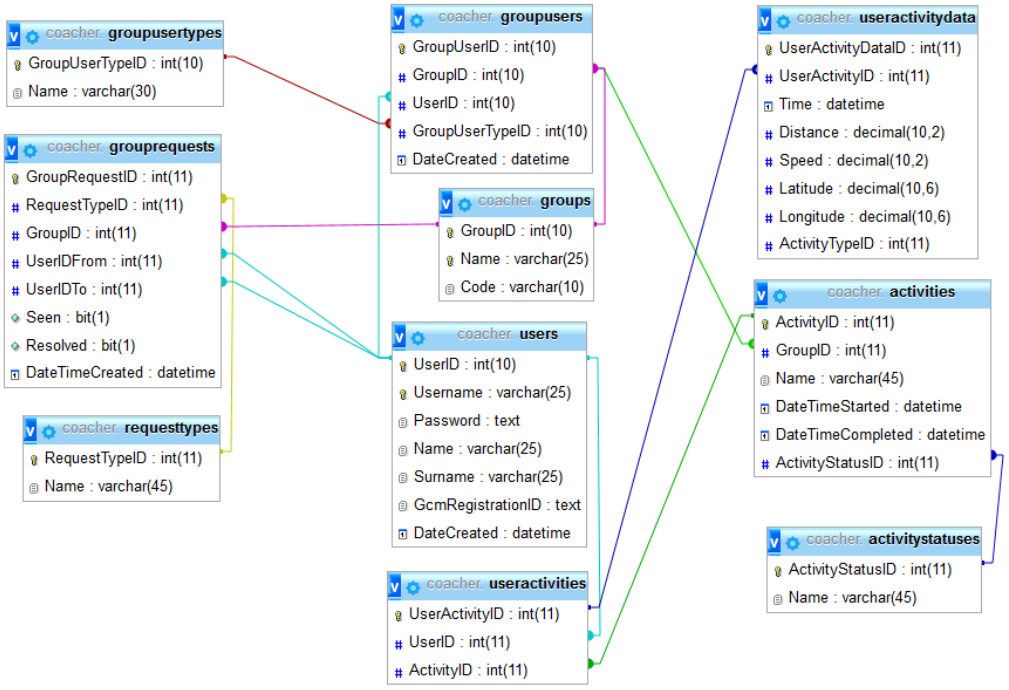
4.1.1 Podatkovni strežnik

Z orodjem XAMPP smo hitro in preprosto ustvarili ustrezno strežniško okolje za naš projekt. Samodejno je poskrbel za namestitev strežnika Apache HTTP in podatkovne baze MySQL, kamor smo kasneje shranili podatke. XAMPP nudi strežniško okolje, ki je namenjeno predvsem razvoju aplikacij in spletnih strani na lokalnem računalniku, zato smo ga morali ustrezno prilagoditi našim razmeram. Privzeto so lahko do strežnika dostopale le naprave iz istega omrežja, zato smo z vpisom ustreznih ukazov v nastavitveno datoteko htppd.config omogočili dostop vsem napravam, ne glede na omrežje, v katerem se nahajajo.

```
<LocationMatch
    "^/(?i:(?:xampp|security|licenses|phpmyadmin|webalizer
server-status|server-info))">
        Allow from all
    ErrorDocument 403 /error/XAMPP_FORBIDDEN.html.var
</LocationMatch>

<Directory "C:/xampp/php">
    AllowOverride AuthConfig Limit
    Order allow,deny
    Allow from all
    Require all granted
</Directory>
```

4.1.2 Podatkovna baza

Relacijska podatkovna baza MySQL, postavljena na našem strežniku, je sestavljena iz tabel in relacij, ki hranijo podatke o uporabnikih, povezavah med njimi in skupinah, v katerih se nahajajo. Baza hrani tudi vse podrobnosti o aktivnostih, ki so jih igralci opravili, tako da lahko kasneje nad njimi izvajamo različne analize. Dostopi do podatkovne baze potekajo izključno prek klicev procedur, ki nad podatki opravljajo tipične operacije, kot so branje, dodajanje, urejanje in brisanje.



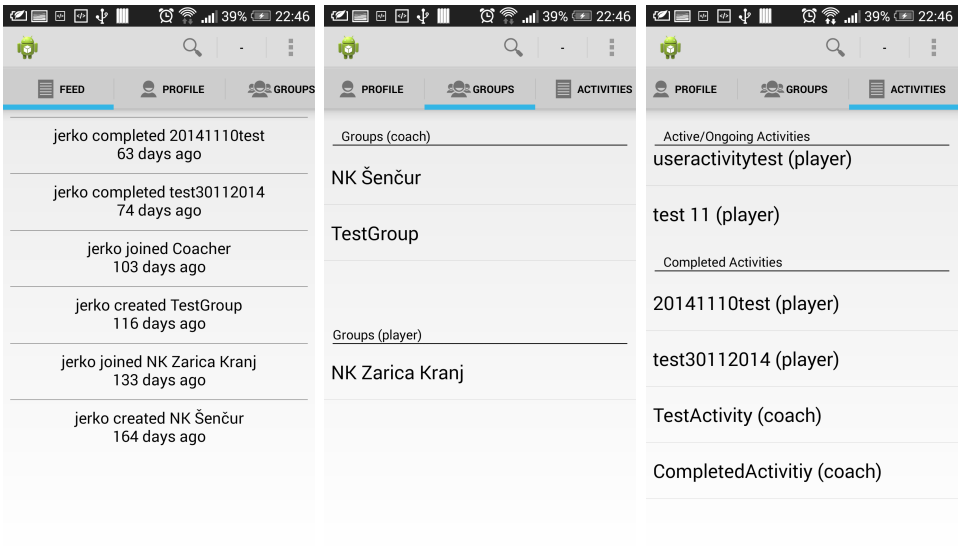
Slika 4.3: Slika podatkovne baze

Tabela users hrani osnovne podatke o uporabnikih, kot so uporabniško ime, geslo ter drugi osebni podatki. Prav tako pa vsebuje ključ RegistrationID, ki je pomemben za prejemanje GCM sporočil. Tabela groups hrani podatke o moštvih, ki so jim prek povezovalne tabele groupusers določeni člani in njihove vloge. Osnovne podatke o treningih, kot so datum in čas začetka treninga, njegov naziv ter status, hranimo v tabeli activities. Igralci,

ki so bili na treningu aktivni, so na aktivnost povezani prek povezovalne tabele useractivities, njihovi podatki, pridobljeni v aplikaciji pa se shranjujejo v tablo useractivitydata. Prošnje in povabila za v skupino se hranijo v tabeli grouprequests.

4.2 Mobilna aplikacija

Razvoj aplikacije smo ločili na tri nivoje, ki imajo specifično vlogo. Najnižji nivo skrbi za povezavo do podatkovne baze in pridobivanje podatkov prek klicev procedur. Pridobljene podatke nato v naslednjem nivoju preslikamo v entitete, ki jih v zadnjem koraku prikažemo v aktivnostih.



(a) Dejavnost (b) Skupine (c) Seznam treningov

Slika 4.4: Domači pogled aplikacije

4.2.1 Povezava aplikacije do podatkovne baze

Naša aplikacija dostopa do podatkovne baze prek JDBC Android API-ja. Za pravilno delovanje API-ja smo morali v naš projekt vključiti knjižnico z ustreznimi metodami in objekti.

Za ustvarjanje, držanje in zapiranje povezave do podatkovne baze poskrbi razred DBConnectionConfig, ki hrani vse potrebne podatke, kot so IP strežnika, odprta vrata, ime podatkovne baze ter uporabniško ime in geslo, potrebna za dostop do nje. Z omenjenimi podatki lahko izvedemo klic, ki ustvari povezavo do podatkovne baze. Povezavo posredujemo ostalim razredom DB na tem nivoju, prek katerih lahko opravimo klic procedur in poizvedujemo po podatkih. Po koncu vseh klicev pa moramo nujno poskrbeti, da se povezava ustrezno zapre.

```
public void SetConnection()
{
   Class.forName("com.mysql.jdbc.Driver");
   Connection connection =
       DriverManager.getConnection("jdbc:mysql://" + DB_IP + ":" +
       DB_PORT + "/" + DB_NAME , DB_USERNAME, DB_PASSWORD);
}
```

4.2.2 Preslikava tabel v entitete

S pomočjo objekta DBConnectionConfig in JDBC API-ja imamo možnost izvajanja poizvedb po naši podatkovni bazi. Razredi, zadolženi za pridobivanje podatkov, vsebujejo metode, ki opravljajo ustrezne klice procedur na podatkovno bazo in vračajo rezultate. Rezultate poizvedb dobimo v obliki objekta ResultSet, ki hrani podatke v parih ime in vrednost. Za lažjo uporabo in boljšo preglednost podatkov pri samem izpisu smo takoj za klicem procedur izvedli preslikavo pridobljenih vrednosti v entitete, ki ohranjajo podobno strukturo kot tabele v podatkovni bazi.[15]

```
public User userInfo(int userID)
{
   ResultSet rs = null;
   User user = null;
   CallableStatement cs = connection.prepareCall("{call
       sp_user_info(?)}");
   cs.setInt(1, userID);
   rs = cs.executeQuery();
   while (rs.next())
   {
      String username = rs.getString("Username");
      String name = rs.getString("Name");
      String surname = rs.getString("Surname");
      user = new User(userID, surname, name, username);
   }
      rs.close();
      return user;
}

public User(int uID, String n, String s, String u)
{
   UserID = uID;
   Name = n;
   Surname = s;
   Username = u;
}
```

4.2.3 Pridobivanje podatkov

Glavni del vsake aplikacije Android so aktivnosti, ki predstavljajo en zaslon aplikacije z uporabniškim vmesnikom. Prek klicev zunanjih storitev aktivnosti podatke pridobivajo, obdelujejo in v zadnjem koraku tudi prikažejo. Ob zagonu aktivnosti se prične izvajati glavna nit aplikacije, ki skrbi za izris uporabniškega vmesnika in reakcije na uporabnikove ukaze in geste. Ker se programski ukazi izvajajo zaporedno, povzročijo dostopi do podatkovnih baz ter izvajanje kompleksnih in zahtevnih operacij neodzivnost programa. Ob koncu metode onCreate trenutne aktivnosti pokličemo asinhroni razred AsyncTask, ki se izvede v drugi niti programa in deluje vzporedno z glavno. V metodi doInBackground razreda AsyncTask se po prej omenjenem načinu ustvari povezava do podatkovne baze in izvede ustrezna poizvedba. Ti podatki se preslikajo v entitete, ki jih trenutna aktivnost potrebuje za prikaz in obdelavo.

```
class AttemptConnectHome extends AsyncTask<String, String, String>
    {
   protected String doInBackground(String... args) {
      User user = null;
      dbConnectionConfig.SetConnection();
      DBProfile dbProfile = new
          DBProfile(dbConnectionConfig.getConnection());

      user = dbProfile.userInfo(userID);
      return "";
   }
}
```

Pridobivanje informacij o uporabnikovem položaju

Glavni del aplikacije predstavlja pridobivanje informacij o trenutnem gibanju in položaju igralcev prek Location API-jev ter njihovo prikazovanje. Location API-ji so del storitve Google Play, ki jih moramo za pravilno delovanje ustrezno nasloviti. Poleg zahtevka za delovanje storitev Google Play moramo od naprave zahtevati tudi dovoljenja za dostop do njenega trenutnega položaja. Dostop do Location API-jev poteka prek Google API Clienta. Po uspešni povezavi prek Google API Clienta lahko pričnemo z zbiranjem podatkov, tako da od storitev Location v intervalih zahtevamo informacije o našem položaju.[16]

```
<uses-permission
    android:name="android.permission.ACCESS_FINE_LOCATION" />

googleApiClient = new GoogleApiClient.Builder(this)
        .addApi(LocationServices.API)
        .addConnectionCallbacks(this)
        .addOnConnectionFailedListener(this)
        .build();

googleApiClient.connect();

public void onConnected(Bundle arg0) {
        locationRequest = LocationRequest.create();
        locationRequest.setPriority(
         LocationRequest.PRIORITY_HIGH_ACCURACY);
        locationRequest.setInterval(SECONDS * 1000);
        LocationServices.FusedLocationApi.requestLocationUpdates(
         googleApiClient, locationRequest,
            ActivityPlayerActivity.this);
}
googleApiClient.disconnect();
```

Igralci, prisotni na treningu, lahko s pritiskom na gumb pričnejo z zbiranjem podatkov. S tem se aplikacija prek Google API Clienta poveže na storitve Location in sproži zahtevo po pridobivanju informacij o položaju igralca in njegovi trenutni hitrosti. Zahteve lahko ustrezno prilagodimo, tako da določimo dolžino intervala in natančnost posodabljanja. Pridobljene podatke zberemo v ustrezno entiteto ter jih shranimo na podatkovni strežnik. Istočasno pa se ti podatki v aplikaciji obdelujejo in računajo pomembne indikatorje igralčevih aktivnosti, kot so povprečna in najvišja hitrost ter skupna pretečena razdalja. Po koncu treninga moramo poskrbeti, da se prejemanje podatkov s storitev preneha, povezava z odjemalcem Google API pa prekine.

Rezultate zahtevkov pridobivamo v metodi onLocationChanged glede na interval, ki smo ga določili. Pridobljene podatke pregledamo, preverimo njihovo natančnost, primerjamo s prejšnjimi ter shranimo na podatkovni strežnik.

```
public void onLocationChanged(Location location) {
   if(isBetterLocation(location, currentBestLocation))
   {
      currLatitude = location.getLatitude();
      currLongitude = location.getLongitude();
      currSpeed = location.getSpeed();

      if(currSpeed > topSpeed)
         topSpeed = currSpeed;

      avgSpeed = totalSpeed / listOfSpeed.size();
      Location.distanceBetween(startLat, startLon, endLat, endLon,
          results);
      currDistance = results[0];
      totalDistance += currDistance;
      currentBestLocation = location;
      new AttemptSaveUserActivityData().execute();
}
```

Prepoznavanje uporabnikovega gibanja

Poleg pridobivanja informacij o položaju, pretečene razdalje in trenutne hitrosti uporabnika, aplikacija prek storitev Location omogoča tudi prepoznavanje gibanja. ActivityRecognition API hitro in učinkovito uporabniku vrača informacije o gibanju naprave brez dodatne uporabe orodij, kot je Accelerometer. Prepoznavanje gibanja smo v našo aplikacijo uvedli podobno kot Location API-je, dodatno smo morali implementirati le odjemalca in storitev, ki skrbita za povezavo ter pridobivanje in obdelavo podatkov. Podobno kot pri ostalih storitvah smo morali v aplikaciji najprej pridobiti ustrezna dovoljenja.[17]

```
<uses-permission android:name=
   "com.google.android.gms.permission.ACTIVITY_RECOGNITION"/>
```

V aplikaciji smo uvedli dodatna razreda DetectionRequester in DetectionRemover, prek katerih smo poskrbeli za povezovanje in prekinjanje povezave do storitev. V obeh razredih smo uporabili objekt ActivityRecognitionClient, ki omogoča generiranje zahtev po prejemanju podatkov o prepoznavanju gibanja. Rezultate zahtevkov pa smo z ActivityRecognitionClient in pomočjo PendingIntent poslali naši storitvi ActivityRecognitionService. Ta je poskrbela za prejem in obdelavo podatkov ter posredovanje rezultatov ustrezni aktivnosti, ki je te podatke uporabila, prikazala in shranila v podatkovno bazo. Pred samim začetkom povezovanja s storitvijo smo morali v aktivnosti omogočiti prejemanje podatkov. Definirali smo objekt BroadcastReceiver in IntentFilter, prek katerega je naša storitev vedela, komu predati prejete rezultate. Po koncu prejemanja podatkov smo poskrbeli za prekinitev povezave s storitvijo prek razreda DetectionRemover.

V aktivnosti, ki skrbi za prejemanje podatkov o gibanju igralca in njegovem položaju, smo definirali objekta DetectionRequester in DetectionRemover.

```
detectionRequester = new DetectionRequester(this);
detectionRemover = new DetectionRemover(this);
```

Ob aktivaciji prejemanja podatkov se je igralcu na treningu aktiviralo tudi povezovanje z ActivityRecognition API-jem prek objekta detectionRequester.

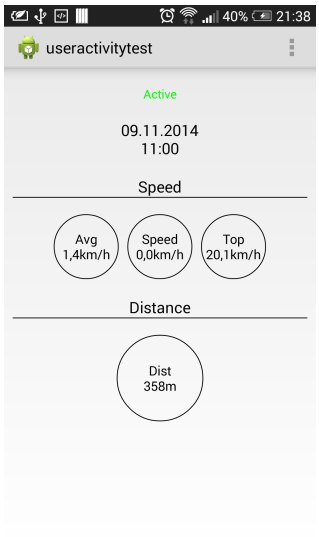
```
detectionRequester.requestUpdates();
```

Pri zahtevku prejemanja podatkov smo morali poskrbeti za inicializacijo objekta ActivityRecognitionClient, prek katerega smo se povezali s storitvijo in od nje zahtevali podatke. Nastavili smo dolžino intervala zahtevkov in določili storitev ActivityRecognitionService, ki smo ji prek objekta PendingIntent prejete podatke posredovali.

```
activityRecognitionClient = new ActivityRecognitionClient(context,
    this, this);

activityRecognitionClient.connect();

public void onConnected(Bundle arg0) {
    activityRecognitionClient.requestActivityUpdates( 1000,
    pendingIntent);
}
Intent intent = new Intent(context,
    ActivityRecognitionService.class);

PendingIntent pendingIntent = PendingIntent.getService(context, 0,
    intent, PendingIntent.FLAG_UPDATE_CURRENT);
```

ActivityRecognitionService v metodi onHandleIntent prejme posredovane podatke, iz njih izlušči najpomembnejše in jih pošlje ustrezni aktivnosti.

```
protected void onHandleIntent(Intent intent) {
   if(ActivityRecognitionResult.hasResult(intent)){
      ActivityRecognitionResult result =
          ActivityRecognitionResult.extractResult(intent);
      Intent i = new
          Intent("com.jerko.coacher.ACTIVITY_RECOGNITION_DATA");
      i.putExtra("ActivityTypeID",
          result.getMostProbableActivity().getType());
      i.putExtra("Activity",
          getType(result.getMostProbableActivity().getType()) );
      i.putExtra("Confidence",
          result.getMostProbableActivity().getConfidence());
      sendBroadcast(i);
   }
}
```



Slika 4.5: Trenutno gibanje igralca

V aktivnosti ActivityPlayerActivity, ki je zadolžena za prejemanje podatkov o uporabnikovem gibanjem, smo poskrbeli še za aktivacijo objekta BroadcastReceiver, ki omogoča aktivnosti prejemanje podatkov, ki jih posreduje storitev ActivityRecognitionService. Po končanem treningu in prekinitvi prejemanja podatkov smo poskrbeli še za ukinitev povezave.

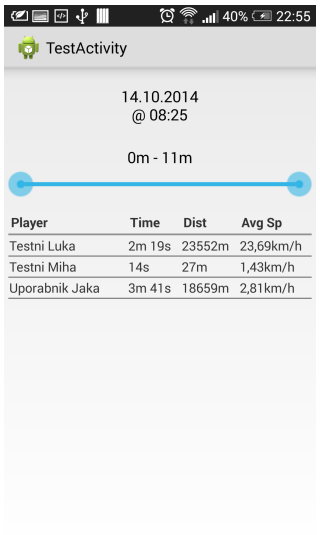
```
broadcastReceiver = new BroadcastReceiver() {
   public void onReceive(Context context, Intent intent) {
      String v = "Activity :" + intent.getStringExtra("Activity") +
          " " +            "Confidence : " +
          intent.getExtras().getInt("Confidence" + "\n";
   currentActivityTypeID = intent.getIntExtra("ActivityTypeID", 4);
   }
};
IntentFilter filter = new IntentFilter();
filter.addAction("com.jerko.coacher.ACTIVITY_RECOGNITION_DATA");
registerReceiver(broadcastReceiver, filter);

detectionRemover.removeUpdates(
   detectionRequester.getRequestPendingIntent());
```

Prikazovanje trenutnega gibanja igralcev

Podatki o gibanju igralca, pridobljeni med treningom moštva, se v vsakem intervalu shranijo v podatkovno bazo. Aplikacija omogoča trenerjem nadzor nad potekom treninga in takojšen pregled nad shranjenimi podatki. Ti se v določenem intervalu s strežnika preberejo in prikažejo v aplikaciji.

V aktivnosti ActivityCoachActivity smo definirali objekt Timer, ki v danem intervalu pokliče ustrezne razrede, ki aktivnosti priskrbi najnovejše podatke trenutnega gibanja igralcev na treningu.

```
final Handler handler = new Handler();
Timer timer = new Timer();
TimerTask doAsynchronousTask = new TimerTask() {
   @Override
   public void run() {
      handler.post(new Runnable() {
         public void run() {
            try {
               attemptGetUserActivityDataTask = new
                   AttemptGetUserActivityData();
               attemptGetUserActivityDataTask.execute();
   ...
};
timer.schedule(doAsynchronousTask, 0, SECONDS * 1000);
```



Slika 4.6: Pregled trenutnega gibanja igralcev

Aktivnost ActivityCoachActivity v tabeli prikazuje igralce in trenutne informacije o njihovem gibanju (trenutna hitrost, skupna pretečena razdalja in pretečen čas). Ti podatki se v izbranem intervalu posodabljajo. Klik na igralca pa vrne aktivnost ActivityPlayerCompleted, ki prikazuje podrobnejše informacije in grafe o aktivnosti izbranega igralca od začetka treninga do tega trenutka.

4.2.4 Obveščanje igralcev prek GCM

Komunikacijo med trenerjem in igralci moštva smo implementirali s pomočjo Google Cloud Messaging. Trenerjem smo omogočili možnost obveščanja igralcev o začetku in koncu treninga, sprememb v načrtu ter dajanje napotkov igralcem med samim treningom.

Za pravilno delovanje storitve GCM smo morali ustvariti nov projekt Google. V projekt smo vključili storitev Google Cloud Messaging API za Android ter si zapisali številko projekta.

Številka projekta omogoča igralcem registracijo na strežnik GCM in prejemanje poslanih sporočil. Pri registraciji uporabnikov v aplikacijo se opravi klic, ki s pomočjo številke projekta uporabniku priskrbi RegistrationID ter si ga zapiše v podatkovno bazo. Strežnik GCM prek vrednosti RegistrationID ugotovi, katerim napravam naj pošlje določeno obvestilo.

Na drugi strani trener za pošiljanje sporočil nujno potrebuje avtentikacijski žeton. Dobili smo ga v našem projektu Google s pomočjo Application Package ter določili IP-je, ki jim je omogočeno pošiljanje sporočil. Avtentikacijski ključ smo si shranili na podatkovni strežnik, uporablja pa se v glavi zahteve POST pri pošiljanju sporočil na strežnik GCM.[18]

Za ustrezno prejemanje in pošiljanje sporočil GCM smo v aplikaciji poleg omenjenih ključev od naprave zahtevali še ustrezna dovoljenja.

```
<uses-permission android:name="android.permission.INTERNET "/>
<uses-permission android:name="android.permission.GET_ACCOUNTS" />
<uses-permission android:name="android.permission.WAKE_LOCK" />
<uses-permission
    android:name="com.google.android.c2dm.permission.RECEIVE" />
<permission
    android:name="com.jerko.coacher.permission.C2D_MESSAGE"
    android:protectionLevel="signature" />
<uses-permission
    android:name="com.jerko.coacher.permission.C2D_MESSAGE" />
```

Od naprave smo zahtevali dovoljenje, ki aplikaciji omogoča dostop do interneta ter pošiljanje in prejemanje GCM sporočil. Prav tako smo v aplikaciji zahtevali dovoljenje za dostop do računa Google, brez katerega GCM na napravah z verzijo Android 4.0.4 ali starejšo ne deluje pravilno. Vključili smo tudi dovoljenje, prek katerega lahko aplikacija napravo zbudi in jo drži zbujeno ob prejemu sporočila GCM, ter dovoljenje, ki aplikaciji omogoča registracijo naprave na strežnik GCM in prejemanje njej namenjenih sporočil. Na koncu smo ustvarili tudi lastno dovoljenje, ki je povezalo našo aplikacijo s sporočili GCM, ki jih prejema. S tem smo drugim aplikacijam na napravi preprečili prevzemanje tovrstnih sporočil GCM.

V manifestu smo prav tako definirali prejemnika za sporočila GcmBroadcastReceiver in mu pripisali dovoljenje za sprejem sporočil GCM. Definirali smo tudi storitev GcmMessageHandler, ki se mu prejeta sporočila posredujejo za obdelavo.

```
<receiver  android:name="com.jerko.coacher.GcmBroadcastReceiver"
        android:permission="com.google.android.c2dm.permission.SEND"
           >
   <intent-filter>
      <action android:name="com.google.android.c2dm.intent.RECEIVE"
         />
      <category android:name="com.jerko.coacher" />
   </intent-filter>
</receiver>
<service android:name="com.jerko.coacher.GcmMessageHandler" />
```

Pošiljanje sporočila GCM poteka prek zahtevka HTTP POST. Podatke, ki jih želimo poslati, smo sestavili skupaj v objekt JSON in mu dodali RegistrationID prejemnikov. Nato smo ustvarili zahtevek HTTP POST, ga naslovili na strežnik GCM, v glavo zahtevka zapisali ustrezen avtorizacijski žeton in dodali objekt JSON.

Najprej smo v objekt dataObject dodali ustrezne podatke aktivnosti, ki jih bodo igralci pri prejemu sporočila videli, nato pa pridobili RegistrationID trenutno prisotnih igralcev na treningu in jih shranili v objekt JSONArray. Na koncu smo ustvarili še končni objekt JSON ter mu dodali pripravljene podatke in RegistrationID-je.

```
JSONObject dataObject = new JSONObject();
dataObject.put("message", activity.getName() + " is " +
    activityStatusName);
dataObject.put("title", activity.getName());
dataObject.put("activityID", activity.getActivityID());
dataObject.put("activityStatusID",
    activity.getActivityStatus().getActivityStatusID());

JSONArray arObject=new JSONArray();
for(int i=0;i<GroupUserGCMRegistrationIDs.size();i++)
   arObject.put(GroupUserGCMRegistrationIDs.get(i));

JSONObject topObject=new JSONObject();
topObject.put("registration_ids", arObject);
topObject.put("data", dataObject);
```

Pošiljanje zahtevka poteka prek objekta HttpPost, ki ga izvedemo prek klica HttpClient. Preden smo izvedli klic, smo objektu HttpPost pripeli naslov URL strežnika GCM in avtorizacijski ključ ter dodali podatke in RegistrationID prejemnikov.

```
String jsonString = topObject.toString();
DefaultHttpClient httpClient = new DefaultHttpClient();
HttpPost httpPost = new HttpPost(GCM_SERVICE_URL);
httpPost.setHeader("Authorization", "key=" + AUTH_KEY);
httpPost.setHeader("Content-Type", "application/json");
httpPost.setEntity(new StringEntity(jsonString));
HttpResponse httpResponse = httpClient.execute(httpPost);
HttpEntity httpEntity = httpResponse.getEntity();
String json_str = EntityUtils.toString(httpEntity);
```

Glavna razreda, ki skrbita za prejem in obdelavo sporočil, sta prejemnik GcmBroadcastReceiver in storitev GcmMessageHandler. GcmBroadcastReceiver skrbi za prejem sporočila in obuditev naprave ob prejemu sporočila. Metoda onReceive sprejme sporočilo in ga pošlje storitvi GcmMessageHandler. Ta prejeto sporočilo obdela in poskrbi za ustrezno izvajanje aplikacije glede na rezultat obdelave.[19]

```
public void onReceive(Context context, Intent intent) {
   ComponentName comp = new ComponentName(context.getPackageName(),
   GcmMessageHandler.class.getName());
   startWakefulService(context, (intent.setComponent(comp)));
   setResultCode(Activity.RESULT_OK);
}
```

Razred GcmMessageHandler v metodi onHandleIntent prek objekta GoogleCloudMessaging posredovane podatke prejme in obdela, zbudi napravo ter ustvari obvestilo, ki se uporabniku prikaže.

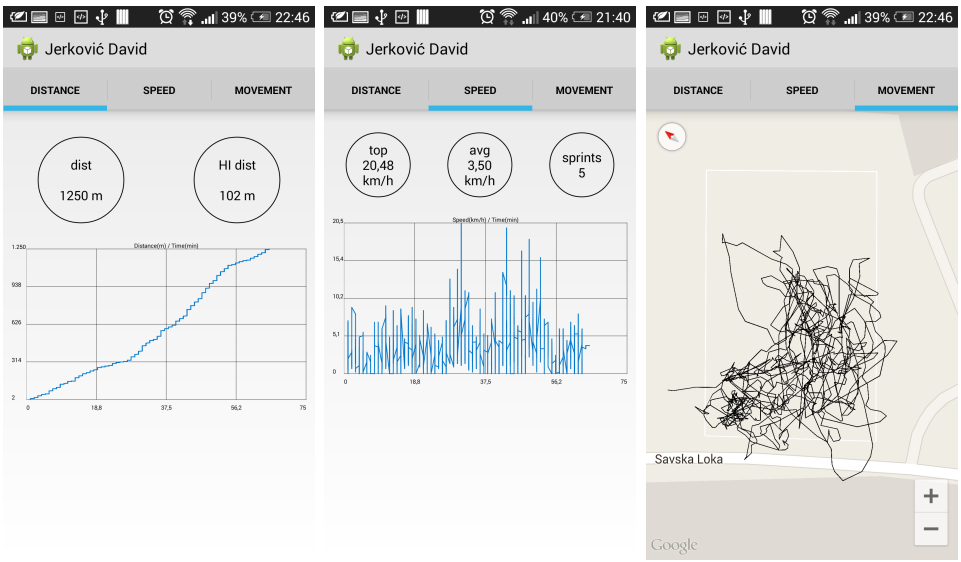
```
protected void onHandleIntent(Intent intent) {
   int mID = 0;
   Bundle extras = intent.getExtras();
   GoogleCloudMessaging gcm =
       GoogleCloudMessaging.getInstance(this);
   String mes = extras.getString("message");
   String title = extras.getString("title");
   String activityID = extras.getString("activityID");
   String activityStatusStringID =
       extras.getString("activityStatusID");
   GcmBroadcastReceiver.completeWakefulIntent(intent);
   NotificationCompat.Builder mBuilder = new
       NotificationCompat.Builder(this)
         .setSmallIcon(R.drawable.ic_launcher)
         .setContentTitle(title).setContentText(mes);
}
```

Na koncu smo prejete podatke obdelali in ugotovili, da se je trening končal. Pripravljenem obvestilu smo zato določili, da se ob kliku nanj aplikacija preusmeri na pregled rezultatov ravnokar končanega treninga.

```
Intent resultIntent = null;

int activityStatusID = Integer.parseInt(activityStatusStringID);
if(activityStatusID == 3)
{
   resultIntent = new Intent(this,
       ActivityPlayerCompletedActivity.class);
    resultIntent.putExtra("ActivityID",
        Integer.parseInt(activityID));
    resultIntent.putExtra("ActivityName", title);
}

TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(ActivityProfile.class);
stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
    stackBuilder.getPendingIntent(0,
    PendingIntent.FLAG_UPDATE_CURRENT
);
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager = (NotificationManager)
    getSystemService(Context.NOTIFICATION_SERVICE);
```

4.2.5 Prikaz rezultatov

Prikaz rezultatov treninga izbranega igralca je sestavljen iz prikaza najpomembnejših kazalcev gibanja, kot so najvišja in povprečna hitrost, pretečena razdalja ter število opravljenih sprintov. Ogledamo si lahko tudi hitrost in pretečeno razdaljo skozi celoten trening v obliki grafov GraphView ter gibanje po igrišču prek GoogleMaps API-ja.



(a) Pretečena razdalja (b) Analiza hitrosti (c) Prikaz gibanja

Slika 4.7: Prikaz pregleda aktivnosti igralca

GraphView

Knjižnica GraphView nam omogoča preprosto in učinkovito ustvarjanje preglednih grafov. V prvem koraku smo zbrane podatke pridobili iz podatkovne baze, jih obdelali in ustrezno pripravili za prikaz. Podatke smo zbrali v polju GraphViewData, ki smo ga prek objekta GraphViewSeries nato posredovali objektu GraphView.

```
GraphViewData[] gvdUserActivityDistance = new
    GraphViewData[userActivityData.size()];

for(int i = 0; i < userActivityData.size(); i++)
{
   totalDistance += userActivityData.get(i).getDistance();
   int time = (int) (userActivityData.get(i).getTime().getTime() /
       1000 / 60 - dateTimeStarted.getTime() / 1000 / 60);
   gvdUserActivityDistance[i] = new GraphViewData(time,
       totalDistance);
}

GraphViewSeries gvsDistance = new
    GraphViewSeries(gvdUserActivityDistance);
gvDistance.addSeries(gvsDistance);
```

Preden smo graf prikazali, smo ga ustrezno prilagodili našim potrebam. Postavili smo mu ime ter določili razne podrobnosti, kot so tip grafa, velikost besedila, legendo, barvo črt in ozadja ter začetno in končno vrednost prikaza podatkov grafa. GraphView prav tako samodejno implementira odziv na uporabnikove geste, kot so povečava in drsenje po grafu.

```
GraphView gvDistance = new
    LineGraphView(ActivityPlayerCompletedActivity.this,
    "Distance(m) / Time(min)");
int dtStarted = (int) ((selectedDateTimeStarted / 1000) / 60);
int dtCompleted = (int) ((selectedDateTimeCompleted / 1000) / 60);
gvDistance.setViewPort(dtStarted, dtCompleted - dtStarted);
gvDistance.setId(1);
```

```
gvDistance.getGraphViewStyle().setGridColor(Color.BLACK);
gvDistance.getGraphViewStyle().setNumHorizontalLabels(labelNum);
gvDistance.getGraphViewStyle().setNumVerticalLabels(labelNum);
gvDistance.getGraphViewStyle().setGridColor(Color.BLACK);
gvDistance.getGraphViewStyle().setTextSize(20);
gvDistance.setFocusable(true);
gvDistance.setScalable(true);
```

Google Maps Android API v2

Pri pridobivanju podatkov prek storitve Location so si uporabniki med drugim shranili tudi geografsko širino in dolžino trenutnega položaja. Prav tako pa smo si v podatkovno bazo shranili kooordinate igrišča, kjer se je trening izvedel. Prek Google Maps API-ja je pri pregledu rezultatov treninga skupaj z igriščem in pridobljenimi podatki prikazano celotno gibanje igralcev med treningom.

Google Maps API je prav tako kot ostali API-ji del storitev Google Play, tako da naša aplikacija že vsebuje potrebno referenco. Za pravilno delovanje smo morali priskrbeti tudi ustrezen ključ Google Maps API. Podobno kot pri GCM API-ju smo v našem projektu Google ustvarili novega za naprave Android in vključili storitev Google Maps API v2. Ključ je vezan na aplikacijo, ne na uporabnike, zato smo morali za pridobitev ključa navesti Aplication Package ter podati certifikat aplikacije v obliki podpisa SHA-1.

```
<meta-data  android:name="com.google.android.gms.version"
           android:value="@integer/google_play_services_version" />
<meta-data
        android:name="com.google.android.maps.v2.API_KEY"
        android:value="AIzaSyB9VBBNgF71UrFSCdnSzvPvgxUX-bO2nis" />
```

Poleg ključa smo morali v aplikaciji od naprave zahtevati tudi ustrezna dovoljenja. Spodnja dovoljenja nam omogočajo preverjanje povezave z omrežjem, prenos zemljevida s strežnikov Google Maps ter shranjevanje zemljevida v pomnilnik telefona za poznejšo uporabo. Prav tako pa smo poleg ustreznih dovoljenj napravi dodali še zahtevo za OpenGL ES verzija 2, ki aplikaciji omogoča sam izris zemljevida.[19]

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission
    android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
    android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
<uses-feature android:glEsVersion="0x00020000"
    android:required="true" />
```

Prikaz zemljevida v aktivnosti aplikacije omogoča element MapFragment, ki poleg zemljevida omogoča tudi izris raznih objektov v njem.

Najprej smo iz podatkovne baze pridobili koordinate igrišča ter ga na zemljevidu prikazali. Pri izrisu smo uporabili objekt Polygon, ki smo mu določili oglišča, ta pa je prek njih izrisal geografski lik.

```
PolygonOptions poPlayingField =
   new PolygonOptions().add(
      new LatLng(pointA[0], pointA[1]),
      new LatLng(pointB[0], pointB[1]),
      new LatLng(pointC[0], pointC[1]),
      new LatLng(pointD[0], pointD[1]),
      new LatLng(pointA[0], pointA[1]));
Polygon polygon = googleMap.addPolygon(poPlayingField);
```

Pri samem prikazu gibanja igralca med treningom smo uporabili posamezne zapise geografske širine in dolžine ter jih združili v objekt PolyLine, ki skozi posamezne točke izriše pot. Najprej smo imeli namen izrisati vse točke posamezno, a se je to izkazalo za potratno operacijo, ki je povzročila zastoj aplikacije.

```
PolylineOptions poMovement = new PolylineOptions();
for(int i = 0; i < userActivityData.size(); i++)
{
   poMovement.add(new LatLng(userActivityData.get(i).getLatitude(),
                             userActivityData.get(i).getLongitude()));
}
poMovement.width(2).color(Color.GRAY);

Polyline polyline = googleMap.addPolyline(poMovement);
```

Google Maps API nam omogoča tudi dodatne možnosti pri prikazu zemljevida, kot so barva črt, poimenovanje krajev, povečavo in premikanje po zemljevidu ter njegovo sukanje.

Poglavje 5

Sklepne ugotovitve

V diplomskem delu smo postavili testni strežnik s pomočjo orodja XAMPP, zgradili podatkovno bazo MySQL in razvili Android aplikacijo, namenjeno organizaciji ekipe, pripravi treningov ter analiziranju učinkovitosti igralcev pri treniranju. Za pridobivanje podatkov med aktivnostjo igralcev smo uporabili orodja in storitve, ki jih ponuja Google. Spremljanje gibanja, pridobivanje trenutne lokacije in hitrosti ter računanje celotne pretečene razdalje so nam omogočale storitve Google Location in ActivityRecognition. Za prikaz rezultatov in raznih analiz smo poleg preprostega izpisa uporabili GraphView, ki je poskrbel za preprost prikaz preglednic ter storitev Google Maps, prek katere smo lahko izrisali gibanje igralcev po terenu. Na koncu smo omogočili tudi preprosto komunikacijo s sporočili med trenerjem in igralci prek Google Cloud Messaging API-ja.

Sam razvoj aplikacije je potekal dokaj tekoče, imeli smo le manjše preglavice. Programiranja za platformo Android smo se lotili s preprostim predznanjem, tako da smo obnovili osnovne koncepte aktivnosti, povezovanje med njimi ter prikazovanje podatkov in prilagajanje uporabniškega vmesnika. Prek poučnih primerov na uradnih Googlovih straneh za razvijalce smo dodobra spoznali in preučili nekatere storitve, ki jih Google ponuja, ter uspešno povezali našo aplikacijo z njimi. Prav tako pa smo se naučili v naš projekt vključiti zunanje knjižnice, s katerimi si lahko bistveno olajšamo delo.

Načrtovanje in implementacija MySQL podatkovne baze ter pisanje ustreznih poizvedb sta potekala brez težav, medtem ko sta postavitev ustreznega okolja in ustvarjanje povezave naših naprav s strežnikom povzročala malce več preglavic. Težave je povzročalo predvsem povezovanje naše naprave izven domačega omrežja s strežnikom. To nam je kasneje uspelo rešiti s pravilnim odpiranjem ustreznih vrat v nastavitvah našega omrežja ter pisanjem ustreznih ukazov v konfiguracijske datoteke našega okolja, ki so omogočila dostop naprav s poljubnega naslova IP.

Aplikacijo bi lahko v prihodnosti izboljšali in razširili. Ena od najpomembnejših izboljšav bi bil lahko dodatni strežnik, na katerega bi shranjevali podatke o aktivnostih igralcev na treningu. V naši rešitvi se ti podatki shranjujejo in posodabljajo vsako sekundo, kar lahko za dolge in pogoste aktivnosti z veliko igralci v naši podatkovni bazi povzroči ogromno število vrstic v tabeli.

Ustrezno rešitev za omenjeni problem predstavlja najnovejša platforma Google Fit, ki je izšla konec oktobra 2014, nekaj mesecev po koncu razvoja naše aplikacije. Gre za sistem, ki združuje shranjevanje podatkov uporabnika, pridobljenih pri raznih športnih aktivnostih, kasneje pa mu nudi tudi dostop do njih. Platforma Google Fit omogoča pridobivanje podatkov prek raznih senzorjev telefona, kot tudi prek naprav z nizkoenergijskim signalom Bluetooth. Pridobljeni podatki se shranjujejo na strežnik Google Fitness Store, dostop do njih pa nam omogočajo Google Fit API-ji. Podatki na strežniku so dostopni mobilnim aplikacijam prek Android API-jev, aplikacijam na drugih napravah pa preko REST API-jev. Z implementacijo platforme Google Fit bi v naši aplikaciji nadomestili uporabo storitev Location in ActivityRecognition za pridobivanje podatkov, kot so trenutna, povprečna in najvišja hitrost, pretečena radalja ter tip gibanja. Prav tako pa bi prek Android API-jev platforme shranjevanje prejetih podatkov preusmerili na strežnike Google Fit Store, naš podatkovni strežnik pa bi s tem poleg informacij o uporabnikih in skupinah vseboval le osnovne podatke o aktivnostih.

Drugo rešitev predstavljajo podatkovne baze NoSQL, ki omogočajo shranjevanje ogromnega števila podatkov. Te nudijo tudi učinkovito poizvedovanje po njih prek ustreznega ključa (v našem primeru igralec/aktivnost). S tem bi shranjevanje na našem podatkovnem strežniku ločili na dva dela, kjer bi strežnik s podatkovno bazo MySQL vseboval osnovne podatke o uporabnikih, skupinah in aktivnostih, podatkovni strežnik NoSQL pa bi shranjeval podrobnejše informacije o aktivnosti igralca na treningu.

Z daljšim testiranjem aplikacije bi lahko glede na vrsto povezave naprave ter močjo signala ustrezno optimizirali tudi dolžino intervala, v katerem se podatki na treningih pridobivajo in shranjujejo.

Namen naše aplikacije je bil ustvariti nizkocenovni informacijski sistem, specializiran za organizacijo ekip v moštvenih športih, ki bi deloval neodvisno od drugih naprav. Za ekipe, ki imajo na razpolago dodatna orodja za merjenje učinkovitosti igralcev na treningih, bi lahko v naši aplikaciji omogočili povezavo z njimi. S prihodom Androida 4.3 lahko aplikacijam omogočimo povezavo z napravami, ki oddajajo nizkoenergijski signal Bluetooth. S tem bi lahko pridobivali podatke z naprav, kot so pasovi za merjenje srčnega utripa, ter shranjevali in prikazovali tudi te podatke.

V prihodnosti bi v naši aplikaciji ustvarili tudi različne uporabniške vmesnike, s katerimi bi lahko trenerji načrtovali taktiko ter prikazali razne segmente treningov. Te načrte pa bi lahko z igralci kasneje tudi delili.

Literatura

[1] MySQL. [Online]. Dosegljivo: http://en.wikipedia.org/wiki/MySQL. [Dostopano Junij 2014].

[2] MySQL Workbench. [Online]. Dosegljivo: http://en.wikipedia.org/wiki/My_SQLWorkbench. [Dostopano Junij 2014].

[3] XAMPP. [Online]. Dosegljivo: http://en.wikipedia.org/wiki/XAMPP. [Dostopano Junij 2014].

[4] Operacijski sistem Android. [Online]. Dosegljivo: http://en.wikipedia.org/wiki/Android_(operating_system). [Dostopano November 2014].

[5] Z. Mednieks, L. Dornin, G. B. Meike, M. Nakamura. *Programming Android*. O'Reilly Media, 2012.

[6] Android SDK. [Online]. Dosegljivo: http://en.wikipedia.org/wiki/Android_software_development#Android_SDK. [Dostopano Junij 2014].

[7] Eclipse (software). [Online]. Dosegljivo: http://en.wikipedia.org/wiki/Eclipse_(software). [Dostopano Junij 2014].

[8] Android Developer Tools. [Online]. Dosegljivo: http://developer.android.com/tools/help/adt.html. [Dostopano Junij 2014].

[9] Google Play Services. [Online]. Dosegljivo: https://developer.android.com/google/play-services/index.html. [Dostopano Julij 2014].

[10] Google Cloud Messaging Overview. [Online]. Dosegljivo: https://developer.android.com/google/gcm/gcm.html. [Dostopano Avgust 2014].

[11] Making Your App Location-Aware. [Online]. Dosegljivo: https://developer.android.com/training/location/index.html. [Dostopano Julij 2014].

[12] Google Maps Android API v2. [Online]. Dosegljivo: https://developers.google.com/maps/documentation/android/start. [Dostopano September 2014].

[13] GraphView. [Online]. Dosegljivo: http://www.android-graphview.org/#features. September 2014

[14] JDBC Overview. [Online]. Dosegljivo: http://www.oracle.com/technetwork/java/overview-141217.html. [Dostopano Julij 2014].

[15] J. Friesen. *Learn Java for Android Development*. Apress, 2014.

[16] Receiving Location Updates. [Online]. Dosegljivo: http://developer.android.com/training/location/receive-location-updates.html. [Dostopano Avgust 2014].

[17] Activity Recognition. [Online]. Dosegljivo: http://developer.android.com/training/location/activity-recognition.html. [Dostopano September 2014].

[18] Implementing GCM Client. [Online]. Dosegljivo: https://developer.android.com/google/gcm/client.html. [Dostopano September 2014].

[19] R. Schwarz, P. Dutson, J. Steele, N. To. *The Android Developer's Cookbook: Building Applications with the Android SDK*. Addison-Wesley, 2013.